Ideograph를 이용한 최적화 및 병렬성 정보 표현에 관한 연구

정성옥* · 고광만**

A Study on the Optimization and Parallelism Information Representation using Ideograph

요 약

최적화란 비효율적인 코드를 구분해 내서 실행 속도 및 기억 공간의 효율성을 높여 주는 방법으로 컴파일러의 각 단계에서 수행된다. Augustus K. Uht에 의해 제안된 Ideograph는 입력 프로그램에 대한 제어 의존성과 자료 의존성에 관한 정보를 동시에 표현할 수 있어 코드 최적화 단계에서 효과적으로 이용될 수 있으며 프로그램에 존재하는 병렬성을 표현하는데 효과적이다.

구문 트리는 원시 프로그램을 정보를 효율적으로 표현할 수 있는 중간 표현으로서 컴파일러 구현에 널리 사용되고 있다. 본 논문에서는 원시 프로그램의 중간 표현이 구문 트리를 입력으로 받아 최적화 정보를 추출한 후 제어 흐름 및 자료 흐름 분석 정보를 추출하여 제어 의존성과 자료 의존성을 Ideograph에 동시에 표현한다.

Abstract

Ideograph is a truly unifies data and procedural dependencies. Ideograph can be used to assist various program optimization, such as common expression elimination, code motion, constant folding etc.

In this paper, we propose an improved representation of the data and control flow dependencies information for the efficient program execution. In pursuing this goal, we propose a model and in particularly implement a dependency information extractor and information table, which contains data and control flow information per a basic block. And then we design and implementation of the optimized abstract syntax tree using Ideograph which has a control flow information and data flow information for source program.

* 광주여자대학교 컴퓨터학부 조교수
** 광주여자대학교 컴퓨터학부 전임강사

1. 서 론

컴파일러는 입력 프로그램에 대해 실행 속도를 개선하고 주기억장치를 효율적으로 사용하기 위해 중간 표현에 대해 최적화를 수행한다. 코드 최적화 단계에서는 입력 프로그램에 대한 중간 표현을 입력으로 받아 중간 표현과 동등한 의미를 유지하면서 코드를 좀더 효율적으로 만들어 코드 실행 시 기억 공간이나 실행 시간을 절약할 수 있는 여러 가지 최적화 동작을 수행한다[3].

코드 최적화는 크게 기계 의존적인 코드 최적화와 기계 독립적인 코드 최적화로 구분된다. 또한 최적화 방법에 따라 흐름 분석 기술(flow analysis technique)을 이용하는 전역 최적화와 부분적인 관점에서 일련의 비효율적인 코드들을 구분해 내고 좀더 효율적인 코드로 수정하는 방법인 부분 최적화로 구분된다[1][5].

트리 형태를 가지고 있는 중간 표현에 대한 최적화의 수행을 고수준 최적화라 부르며 프로그램 상에 존재하는 제어 흐름과 자료 흐름을 쉽게 볼 수 있는 장점을 가지고 있다[1]. Ideograph는 프로그램 상에 존재하는 자료 의존성과 제어 의존성을 동시에 표현할 수 있어 제어 의존성과 자료 의존성을 각각 표현하는 방법에 비해 장점을 가지고 있다[6].

본 논문에서는 원시 프로그램의 중간 표현이 구문 트리를 입력으로 받아 최적화 정보를 추출한 후 제어 흐름 및 자료 흐름 분석 정보를 추출하여 제어 의존성과 자료 의존성을 효과적으로 동시에 표현한다. 이를 위해 Uht에 의해 제안된 Ideograph를 기반으로 하여 변형된 Ideograph를 생성한다.

본 논문의 구성은 2장에서 원시 프로그램에 존재하는 제어 흐름 분석과 자료 흐름 분석 기법에 대해 고찰한다. 제3장에서는 실질적인 Ideograph 를 구성하는 방법에 대해 설명하며 제4장에서는 최적화된 Ideograph 구성 및 병렬성 표현을 위한 기법에 대해 설명한다. 마지막으로 결론과 향후 연구 방향에 대해 기술한다.

2. 제어 및 자료 흐름 분석

2.1 제어 흐름 분석

제어 흐름 분석 단계에서는 프로그램 상에 존재하는 모든 가능한 제어의 흐름을 나타낸다. 제어 흐름 분석 기법은 두 단계의 과정을 거쳐 수행된다. 첫 번째 단계에서는 입력 프로그램에 대해 기본 블록(basic block)을 구성한다. 두 번째 단계에서는 분기(branch)를 고려하여 흐름 그래프를 구성한다[2][3]. 기본 블록은 실행이 시작되어 끝날 때까지 중지되거나 분기되지 않고 기본 블록의 시작점에서 제어가 들어가 기본 블록의 끝에서 제어가 빠져나오는 연속적으로 실행되는 문장들의 집합으로 구성된다. 기본 블록은 리더와 프로그램 코드로서 구성이 되는데 리더는 다음과 같이 정의된다. 첫째, 프로그램이나 프로시주어의 첫 번째 문장을 리더라 하며 둘째, 조건 분기나 무조건 분기문의 목적지가 리더이다. 셋째, 조건 분기나 무조건 분기문의 바로 다음 문장을 리더라 한다[1]. 흐름 그래프란 기본 블록 집합에 제어 흐름에 관한 정보를 추가해서 방향을 가지고 있는 그래프를 말한다. 흐름 그래프에서 노드는 기본 블록을 나타내며 계산을 수행한다. 노드간에는 에지로서 연결되며 제어가 흐르는 방향을 나타낸다[2].

프로그램의 수행에 있어 대부분의 수행 시간은 루프 안에서 소비된다. 루프는 흐름 그래프

안에 있는 노드의 집합이며 노드간에는 강력하게 연결되어 있다. 또한 루프는 하나의 진입점(entry point)을 가지고 있다. 즉, 루프 밖의 임의 노드에서 루프내의 노드에 도달 할 수 있는 유일한 방법은 진입점을 이용한다. 노드 집합 S에서 시작 노드로부터 집합 S에 존재하는 임의의 노드에 도달할 수 있는 모든 경로가 첫 번째로 n 노드를 통할 경우 노드 n을 헤더라 부른다. 또한 흐름 그래프의 모든 경로가 시작 노드로부터 노드 d를 통해서 노드 n에 도달할 수 있다면 노드 d는 노드 n을 지배한다고 하며 노드 d를 지배자라 부른다. 모든 자신 노드는 자신 노드를 지배하며 루프의 시작점은 루프 내에 있는 모든 노드를 지배한다[2][3].

2.2 자료 흐름 분석

코드를 최적화하고 효율적인 코드를 생성하기 위해 컴파일러는 프로그램에 대한 정보를 수집하고 이러한 정보를 흐름 그래프에 존재하는 각 블록에 전해준다. 특히 프로그램상의 자료 흐름 정보는 자료 흐름 분석 기법을 이용한다. 자료 흐름 분석 기법은 자료에 대해서 정의(definition)와 이용(use)으로 간주되는 정보를 제공한다. 자료 흐름에 관한 정보는 프로그램의 여러 시점에서 자료 흐름 정보에 관련이 있는 방정식을 세우고 해결함으로서 수집할 수 있다[1][2][3]. 자료 흐름 문제는 다음과 같이 두 가지 부류로 나누어진다. 첫째, 프로그램의 임의의 점에 제어가 도달하기 전에 어떤 일이 발생할 수 있는가에 관한 문제로서 계산에 영향을 줄 수 있는 부분이며 이것을 전향 흐름이라 한다. 둘째, 프로그램상의 임의의 점에서 제어가 떠난 후에 어떤 문제가 발생할 수 있는가에 관한 문제로서 앞으로 어떤 이용이 영향을 받을 것인가에 관한 문제로서 후향 흐름이라 한다[2][3].

도달 정의는 각 기본 블록 B 와 변수 x에 대해 어떤 문장이 시작점 노드로부터 기본 블록 B에 도달할 수 있는 경로를 이용해서 변수 x를 정의하는 마지막 문장이 될 수 있는가를 결정한다. 자료 흐름 방정식은 자료 흐름 문제에 관해서 방정식을 구성하고 해결하기 위해 두 변수 IN(B), OUT(B)이 있다. 전향 흐름 문제에 대해 블록으로부터 나오는 정보는 블록으로 들어가는 정보에 관한 함수로 정의된다. 도달 정의는 블록 B 안에서 생성되거나 소멸되는 정의 관점에서 함수 f를 정의하는 데 기본 블록 B에 대한 변수 KILL(B)의 의미는 블록 B내의 정의에 의해서 다른 블록에 존재하는 정의가 소멸되는 정의들의 집합이며 변수 GEN(B)는 블록 B내에서 생성되는 정의들의 집합이다. 일반적인 자료 흐름 방정식의 형태는 다음과 같다[2].

$$OUT(B) = GEN(B) \cup (IN(B) - KILL(B))$$

즉, 블록 끝에 존재하는 정보는 블록 안에서 생성되거나 블록 시작점으로 들어가서 소멸되지 않는 정보이다.

3. Ideograph의 구성

3.1 Ideograph의 의미

Ideograph는 자료 의존성과 제어 의존성을 동시에 속성으로 표현할 수 있는 장점을 가지고 있기 때문에 프로그램의 컴파일 시간에 중복 수식 제거, 코드 이동, 상수 폴딩, 루프 최적화 등과 같은 전역 코드 최적화 기법에서 효과적으로 이용된다. 또한 자료 의존성과 제어 의존성 정보를 활용하여 프로그램 상에 존재하는 병렬성

을 효과적으로 표현할 수 있다[6].

Ideograph는 (A, I, F, G) 4개의 튜플로서 정의된다. 튜플 A는 변수의 집합(S), 상수의 집합(C), 제어 가드(Control guard)의 집합(B), 연산자의 집합(O)을 포함하고 있다. 튜플 I는 초기 변수의 집합을 포함하고 있다. 또한 튜플 F는 최종 변수의 집합을 포함하고 있으며 튜플 O는 Ideograph의 구성과 계산을 제어하는 규칙들의 집합이다. 모든 변수는 초기 변수의 집합 안에서 정의되며 임의의 변수 V가 프로그램 상에서 외부 환경으로 출력되거나 외부 변수를 계산하는 데 이용된다면 유용)하다고 한다. 최종 변수의 집합 F에 속하는 유용한 변수(Useful variable)를 직·간접적으로 만들어 낼 수 없는 계산은 쓸모 없는 계산으로 간주되며 무시된다. 또한 각 명령 변수는 v-instance로 불린다.

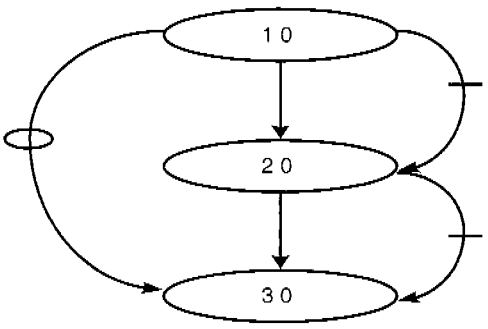
명령 변수 v_i, v_j에 대해서 프로그램 상에서 v_i가 v_j보다 선행된다면 v_j는 v_i에 의존한다고 하며 v_j->v_i로 표기한다. 명령 사이의 의존성은 모든 v-instance에 대한 의존성의 합집합으로 표시한다. 명령 변수인 v-instance v의 계산 결과가 조건 분기 b의 계산 결과에 따라 다르고 v가 유용한 변수라고 하면 명령 변수 v는 조건 분기 b에 대해 제어 의존성을 가지고 있다고 한다.

3.2 Ideograph 구성

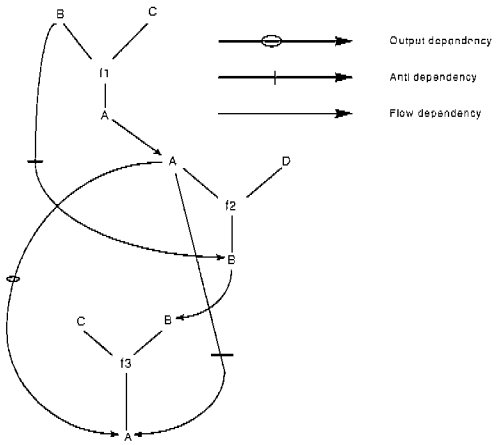
Ideograph의 구성은 첫 번째로 기본 블록을 구성함으로서 시작된다. 기본 블록들은 제어 에지나 흐름 에지 중 하나에 의해 연결된다. 모든 기본 블록과 명령의 식별이 끝났을 때 v-instance의 변수를 이용해서 자료 의존성을 연결한다. 자료 의존성을 연결하는 에지는 명령어 사이를 연결하는 것이 아니라 v-instance 사이를 연결하고 있으며 에지는 제어 연산자나 가드를 레이블로 가지고 있다. 다음은 아래 프로그램 코드에 대한 기본 블록과 명령어들이 구분된 후 명령어들 간에 존재하는 자료 의존성과 v-instance간의 자료 의존성을 그래프 형태를 그림 1과 그림 2에서 보여주고 있다.

여러 가지 코드 최적화를 수행하기 위해 v-instance에 연관된 임시 자료 구조를 가지고 있다. 이런 임시 자료 구조는 최적화 수행을 마친 후에 소멸된다.

```
(10) A := f1(B, C);
(20) B := f2(A, D);
(30) A := f3(C, b);
```



〔그림 1〕 명령어 사이의 자료 의존 그래프



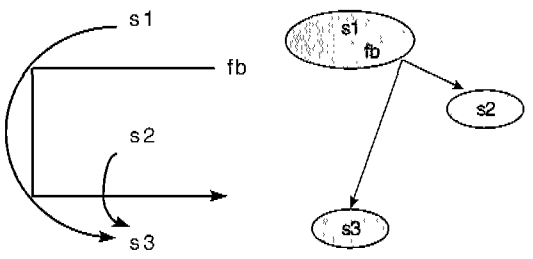
〔그림 2〕 v-instance 사이의 자료 의존 그래프

최적화를 수행하는 동안 v-instance 사이의 자료 의존성의 연결과 유지는 Ultra-Fine-Grain(UFG)이라는 새로운 프로시쥬어에 의해서 수행된다. 자료 의존 그레프를 구성하는 UFG 프로시쥬어는 v-instance사이의 자료 의존성을 유지하고 이용하고 있다. 기본 블록을 이용해서 흐름 그래프를 구성하고 UFG를 이용해서 자료 흐름 그래프를 구성한 후에 모든 조건 분기를 Ideograph 에지의 속성으로 간주되는 제어 가드로 번역한다. 즉, 제어 가드는 조건 분기의 효과를 나타낸다. 이런 프로시쥬어는 자료와 제어 의존성을 하나의 일치된 표기로서 나타낸다. 가드가 UFG 에지에 추가되고 v-instance가 분기의 영향 하에 존재하는지 여부에 따라 참, 거짓 값을 레이블로 갖는다. (v_i, v_j)와 같은 UFG 에지는 v_i에서 v_j에 도달할 수 있는 하나 이상의 직접 실행 경로가 존재한다면 여러 개의 Ideograph의 에지로 변형된다.

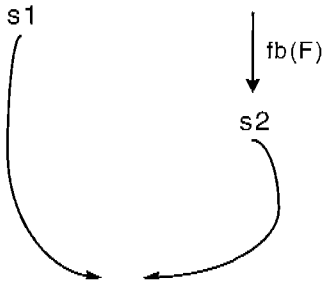
3.3 Ideograph 구성 예

S1 문장을 수행한 후 전향 조건 분기(Forward branch; fb)를 수행하여 참값이면 S3 문장을 수행하고 거짓이면 S2 문장을 수행하는 프로그램 코드에 대한 자료 의존 그래프는 그림 3과 같다.

또한 이런 자료 의존성을 나타내는 그래프를 이용해서 그림 4와 같은 흐름 그래프를 구성한다.



〔그림 3〕 자료 의존 그래프 〔그림 4〕 흐름 그래프



〔그림 5〕 Ideograph 구성 예

문장 S1, S2, S3이 각각 "a := ...", "a := ...", "... := a"와 같은 형태를 가지고 있으며 S2는 S1을 재정의 한다고 가정하는 경우 경로 <S1, S2, S3>에서 자료 의존성을 나타내는 에지 (S1, S2)에 대해서 'a' 는 문장 S2에 의해서 재정의 되고 레이블로 "fb(F)"를 가질 수 없지만 다른 경로 <S1, S3>인 경우 에지 (S1, S3)에 대해서는 "fb(T)" 태그를 갖는 Ideograph를 그림 5에서 보여주고 있다.

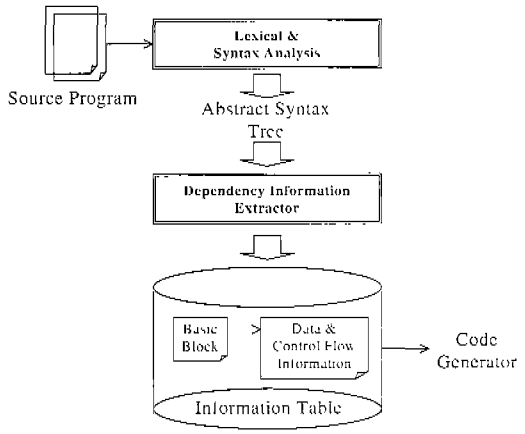
Ideograph 구성에서 중요한 점은 모든 분기가 제어 가드로 변환된 상태인 Ideograph를 구성한 후에 모든 의존성을 나타낼 수 있으며 비로소 최적화를 수행할 수 있다.

4. 최적화된 병렬성 정보 추출시스템

4.1 시스템 모델 구성

C 언어로 작성된 원시 프로그램에 대한 최적화 병렬성 정보 추출하기 그림 6과 같은 시스템 모델을 구성한다.

최적화된 병렬성 정보 추출을 위해 먼저 어휘 분석기와 구문 분석기를 이용하여 원시 프로그램에 대한 구문 트리를 구성한다. 의존성 정보 추출기(dependency information extractor)는 구문 트리



〔그림 6〕 시스템 모델 구성

를 순회하면서 각 기본 블록에 대한 자료 흐름 정보와 제어 흐름 정보를 Ideograph에 표현한다. 이렇게 생성된 정보는 정보 테이블에 저장되어 있으며 특정 목적기계 코드 생성기를 이용하여 목적기계 코드를 생성한다.

4.2 기본 블록 생성

최적화된 구문 트리 생성을 위해 원시 프로그램에 대한 어휘 분석 단계와 구문 분석 단계를 거친 후 예를 들어 bubble() 함수에 대한 기본 블록을 그림 7과 같이 구성한다. 구성된 기본 블록 단위로 자료 흐름 그레프와 제어 흐름 그래프 생성 정보를 Ideograph에 표현하며 보다 최적화된 구문 트리 생성을 위한 정보를 추가한다.

```
bubble()
{
  for( i=1 ; i<n ; i++ )
      for( j=1 ; j>=i ; j++)
        if( a[j]>a[j+1] ) {
          /*  Swapping  */
          }
}
```

기본블록-1	i=1
기본블록-2	i<n
기본블록-3	i++
기본블록-4	j=1
기본블록-5	j>=1
기본블록-6	j++
기본블록-7	a[j] > a[j+1]
기본블록-8	/* ---- */

〔그림 7〕 bubble() 함수에 대한 기본 블록 구성

4.3 최적화 동작

도달 정의로부터 이용-정의 사슬(use-definition chain)을 구성할 수 있다. 즉 변수의 정의와 이용이 연결 리스트로 구성한다. 블록 안에서 정의는 정의를 이용하는 이용과 연결된다. 블록 안에서 정의가 없다면 블록에 도달하는 모든 정의는 이용과 연결된다. 도달 정의는 Ideograph의 구성으로 얻을 수 있으며 이용-정의 사슬은 상수 전달(constant propagation) 최적화 동작에 이용된다.

임의의 수식 계산이 중복되는 경우 먼저 계산된 수식의 값이 변경되지 않았으면 먼저 계산된 수식의 값을 이용함으로서 중복된 수식의 계산을 제거할 수 있다. Ideograph상에서 중복된 수식을 제거하기 위해 다음을 정의한다. 변수 Ck는 임의의 명령 k를 정의하고 변수 Sk는 명령 k의 수식을 정의한다. 또한 변수 Ck에 대한 두 개의 피연산자 v1, v2에 대해 $S_k = f_k(v_k, v_k)$를 정의한다. S_i의 v-instance가 변경되기 전에 프로그램 상에서 수식 S_i에 의해서 재사용 되는지를 발견하는 알고리즘은 아래와 같다.

입력 : R_j, 명령 C_k의 R_j
출력 : $S_j = S_i$ 이거나 $S_j \neq S_i$
방법 : G_u의 원소인 어떤 v_j에 대해
$(R_i \cap R_j) \neq 0$ 이고 $f_i = f_j$ 이면 $s_j = s_i$ 이다

위 알고리즘에서 v-instance vi의 흐름 의존 집합 R_i를 호출하며 피연산자 v_i, v_j에 관한 Si의 두개의 흐름 의존 집합은 각각 R_i와 R_j이다. 또한 v-instance v는 변수 v에 흐름 의존하는 모든 명령을 가지고 있는 집합 G_v를 가지고 있다고 가정한다.

루프내의 문장 수행이 수행될 때마다 같은 값을 갖는 문장(Loop invariant)의 제거는 프로그램 수행 속도를 향상시킨다. Ideograph상에서 이런 값을 갖는 문장은 Ideograph가 순환(iteration) 문장 간에 흐름 의존성을 가지고 있지 않는 특성을 이용해서 코드 최적화시킨다. 상수 폴딩, 코드 이동 등의 기타 다른 최적화 동작은 기존의 방법을 이용하여 최적화 동작을 수행한다.

5. 결 론

입력 언어의 중간 표현에 대해서 코드 최적화의 실행은 코드 실행 시 기억 공간이나 수행 시간을 절약 할 수 있다. 프로그램 상에 존재하는 제어 의존성과 자료 의존성을 각각 나타내야 하는 방법에 비해 Ideograph는 동시에 하나의 표현에 나타낼 수 있는 장점을 가지고 있다. 본 논문에서는 원시 프로그램에 대해 중간 표현으로 구문 트리를 구성한 후 구문 트리를 참조하여 프로그램 상에 존재하는 제어 의존성과 자료 의존성을 동시에 Ideograph에 표현하였다. 또한 Ideograph를 참조하여 중복 수식 제거, 루프 최적화, 코드 이동과 같은 코드 최적화 동작을 수행하였다.

앞으로 보다 효율적인 코드 생성을 위해 더 많은 최적화 기법을 보완할 예정이며 Ideograph 상에서 병렬 요소를 검출하여 병행 수행에 적합한 코드를 생성하도록 연구할 예정이다.

참 고 문 헌

[1] Alfred V. Aho & Ravi Sethi & Jeffrey D. Ullman, Compilers : Principles, Techniques, and Tools, Addison Wesley, 1986.

[2] J. Ferrante & K. Ottenstein & J. Warren, "The program dependence Graph and Its Use in Optimization", ACM TOPLAS, Vol. 9, NO. 3, pp. 319-349, July, 1987.

[3] Karen A. Lemone, Design of Compilers : Techniques of programming language translation, CRC Press, 1992.

[4] S. P. Midkiff & D. A. Pauda, Issues in the Optimization of Parallel Programs, International Conference on Parallel Processing, Vol. 2, pp. 105-113, 1990.

[5] Jean-Paul Tremblay & Paul G. Soreson, Compiler Writing, McGraw-Hill International Editions, 1989.

[6] S. ShouHan Wang & Augustus K. Uta, "Program Optimization with Ideograph", International Conference on Parallel Processing, Vol. 2, pp. 153-159, 1989.

[7] Steven S. Muchnick, Advanced Compiler Design and Implementation, Morgan Kaufmann, 1997.

[8] Tom Axford, Concurrent Programming : Fundamental Techniques for Real-Time and Parallel Software Design, John Wiely & Sons, 1989.

[9] Gregory R. Andrews, Concurrent Programming : Principles and Practice, The Benjamin/Cummings Publishing Company, Inc., 1991.

[10] Tom Axford, Concurrent Programming : Fundamental Techniques for Real-Time and Parallel Software Design, John Wiely & Sons, 1989.